Functional programming

(int -> int) list

Falcon
- * first-class functions: functions are values
- higher-order function: a function that either takes a function as arg or returns a function
- * partial application: pass some args to a fn but not all (apply incrementally)
- * anonymous functions

compile_expression env e

List.map (compile_expression env) exprs

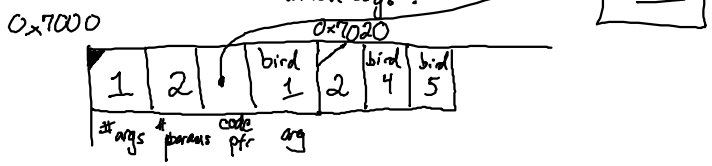
(fun a -> f a b)

```
def inc n =
  n + 1
end
def f g x =
  g (g x)
end
let plus_two = f inc in

f inc 5
```

```
def add x y =
  x + y
end
let inc = add 1 in
let x = (4, 5) in
inc 4 + inc 5
```

- which original function?
- how many params?
- how many args?
- which args?



$2^{60} > 10^{18}$

@ 0x7000 1000

0x8000000000000001 0x0000000000000002 0x0000000004197235 0x0000000000000002

0x0000000000000002 0x0000000000000008 0x000000000000000A

func 3

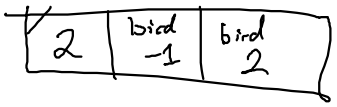
- saturated application: when you have enough args
- oversaturated application: you don't have enough args

If #args+1 < #params Then
1. Copy whole closure to a new location
2. Inc # args
3. Copy new arg into position
4. Result: ptr to new heap object

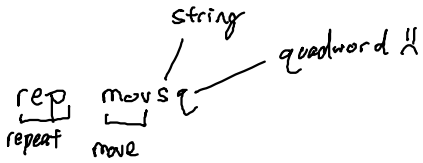
Else:
GO TIME

1. Copy args onto stack
2. Call fn
3. Remove args

(-1, 2)



x86-64:



rdi – destination ptr
rsi – source ptr
rcx – # of things to copy

rep movsq

```
def inc n =
  n+1
end

let add1 = inc in
add1 5
```


```
closure_of_inc:
    dq 0x8000000 0000 0000, 1, fn_inc
```